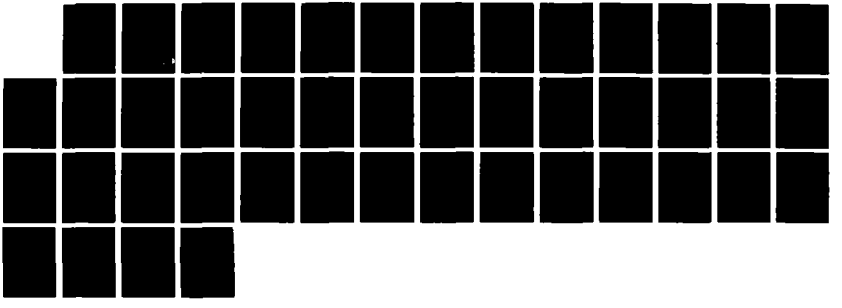
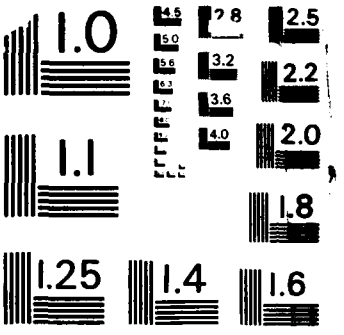
AD-A179 515 INTERMEDIATE LEVEL COMPUTER VISION PROCESSING ALGORITHM 1/1
DEVELOPMENT FOR T..(U) MASSACHUSETTS UNIV AMHERST
RISEMAN 29 NOV 86 AFOSR-TR-87-0453 F49620-86-C-0041
UNCLASSIFIED F/G 9/2 NL





MICROCOPY RESOLUTION TEST CHART

NATIONAL BUREAU OF STANDARDS - 1963 - A

SECURITY CLASSIFICATION OF THIS PAGE

REPORT DOCUMENTATION PAGE

1a. REPORT SECURITY CLASSIFICATION: Unclassified	1b. RESTRICTIVE MARKINGS
2a. SECURITY CLASSIFICATION AUTHORITY	3. DISTRIBUTION/AVAILABILITY OF REPORT: Approved for public releas; distribution unlimited
AD-A179 515	5. MONITORING ORGANIZATION REPORT NUMBER(S): AFOSR-TR- 87-0453
6a. NAME OF PERFORMING ORGANIZATION: University of Massachusetts; 6b. OFFICE SYMBOL (If applicable):	7a. NAME OF MONITORING ORGANIZATION: AFOSR
6c. ADDRESS (City, State and ZIP Code): Amherst, MA 01003	7b. ADDRESS (City, State and ZIP Code): Bldg 410, Bolling AFB, DC 20332-6448
8a. NAME OF FUNDING/SPONSORING ORGANIZATION: Same as 7a; 8b. OFFICE SYMBOL (If applicable): NE	9. PROCUREMENT INSTRUMENT IDENTIFICATION NUMBER: F49620-86-C-0041
8c. ADDRESS (City, State and ZIP Code): Same as 7b	10. SOURCE OF FUNDING NOS.

PROGRAM ELEMENT NO.	PROJECT NO.	TASK NO.	WORK UNIT NO.
61102F	5668	DARPA 00	

11. TITLE (Include Security Classification): Intermediate Level Computer Vision Processing Algorithm Dev. for the Cont

12. PERSONAL AUTHOR(S): Dr. Riseman

13a. TYPE OF REPORT	13b. TIME COVERED	14. DATE OF REPORT (Yr., Mo., Day)	15. PAGE COUNT
R&D Progress	FROM 12/86 TO 02/87	86 Nov. 29	37

16. SUPPLEMENTARY NOTATION

17. COSATI CODES			18. SUBJECT TERMS (Continue on reverse if necessary and identify by block number)
FIELD	GROUP	SUB. GR.	

19. ABSTRACT (Continue on reverse if necessary and identify by block number)

During this quarter we have finalized the low to intermediate level interfac hardward design for the ICA. We have also been examining various network topologies for communications within the intermediate level of the ICA. At the minimum we will support a mesh with an ouverlayed hypercube However, we are looking into a fully programable network topology that will allow us to experiment with different processor interconnection schemes.

20. DISTRIBUTION/AVAILABILITY OF ABSTRACT	21. ABSTRACT SECURITY CLASSIFICATION
UNCLASSIFIED/UNLIMITED ☐ SAME AS RPT. ☐ DTIC USERS ☐	

22a. NAME OF RESPONSIBLE INDIVIDUAL	22b. TELEPHONE NUMBER (Include Area Code)	22c. OFFICE SYMBOL
Dr. C. L. Giles	202-767-4933	NE

DD FORM 1473, 83 APR EDITION OF 1 JAN 73 IS OBSOLETE.

SECURITY CLASSIFICATION OF THIS PAGE

Intermediate Level Computer Vision Processing
Algorithm Development for the
Content Addressable Array Parallel Processor

University of Massachusetts: F49620-86-C-0041

Quarterly Status Report #3, November 29, 1986

During this quarter we have developed and analyzed a set of seven benchmark problems for the IUA. These included Hough Transform, Convex Hull, Voronoi Diagram, Minimal Spanning Tree, Visibility of Vertices in a projected 3-dimensional model, sub-graph isomorphism, and the minimum cost path between points in a weighted graph.

These problems are commonly considered intermediate level procesing in many visions research groups. We have also continued to develop parallel implementations of UMass intermediate level processing algorithms, such as Boldt's line merging and Anandan's motion analysis.

A commercial processor, the TMS320C25, has been chosen as our Intermediate Communications and Associative Processor (ICAP) procesing element. The TMS320C25 has the advantages that it is a five million instruction per second signal processing unit with a fast multiplier and software support for fast floating point operations. It also has a built-in 5 Mb/S serial port that will interface well with the intermediate level communications network.

We have also been exploring a set of group theoretic network topologies with respect to the communication needs of intermediate level processing. This has required us to analyze the classes of communication needed in each of the algorithms we have implemented.

During the next period we will be revising the CAAPP level simulator and transporting

it to the Texas Instruments Explorer Work Station. We will also begin to implement an ICAP simulator on an Explorer that has been augmented with the TMS320C25 processor.

A summary of the fiscal status of the grant is:

Amount currently provided for the contract	$197,000
Expenditures and commitments to date	65,926
Estimate date of completion of work	2/14/88

Intermediate Level Computer Vision Processing Algorithm Development for the Content Addressable Array Parallel Processor

University of Massachusetts: F19620-86-C-00411
Quarterly Status Report No.4

March 1, 1987

During this quarter we have finalized the low to intermediate level interface hardware design for the IUA. We have also been examining various network topologies for communications within the intermediate level of the IUA. At the minimum we will support a mesh with an overlayed hypercube. However, we are looking into a fully programmable network topology that will allow us to experiment with different processor interconnection schemes.

We have also installed our Texas Instruments Explorer workstation with an Odyssey multi-processor board. The Odyssey contains four of the TMS32020 processors that are upward compatible to the TMS320C25 processors that will be used in the intermediate level of the IUA. The Odyssey board will allow us to perform high speed simulations of the low and intermediate levels of the IUA, and will be especially useful in simulating the interactions between those levels.

Because the Odyssey is a beta-test product, we spent a significant amount of time debugging the hardware and software after it was first installed. However, simulator development is now progressing quickly: the low level simulator is nearly debugged and we expect to complete the intermediate level and interface simulators in another month or so.

Once the simulator is completed, we will begin to build the IUA software support and development environment on top of it. We will also re-implement our existing intermediate level algorithms on the Odyssey simulator.

In addition to the Odyssey-based simulator, we are implementing a VAX-based simulator in C that will be transportable to other sites, and which will interface to the UMass VISIONS software environment. At the end of this reporting period, the low level portion of this simulator was complete. Further development will take place in the next quarter.

We re-coded the seven DARPA benchmark tasks (initially implemented in the preceding quarter) to take advantage of various IUA features that were not available

in our old simulator, but which will be present in the hardware. This provided significant improvements in our timings (a copy of the final benchmark result table is attached).

We are also working with DARPA and Azriel Rosenfeld at the U. of Maryland to develop a new set of benchmarks that will test performance on an integrated image interpretation task. This new benchmark will focus on the issue of interaction between the low and intermediate levels of representation or abstraction in computer vision.

A summary of the fiscal status of the grant is:

Amount currently provided for the contract	$197,000
Expenditures and commitments to date	$108,022
Estimated date of completion of work	February 14, 1988

Image Understanding
Benchmark Timing Summary
(all times in seconds)

	Task time (seconds)								
Architecture	(1a-b)	(1c)	(2)	(3)	(4a)	(4b)	(4c)	(5)	(7)
UMass IUA									
Time with									
32 bit floating point	NA	NA	NA	NA	0.015	–	0.124	0.29	0.001
Time with									
16 bit fixed point	0.0002	0.0002	0.00005	0.027	0.007	0.05**	0.072	0.02	0.0007
Butterfly									
16 nodes	18.3			15.2	0.19	—	—	67	–
100+ nodes	2.9	8.2	7.2	7.4		-	—	4.15	-
Non-Von	0.002		1	0.4	0.04	—	0.04	0.1	0.04
Connection Machine	0.003	0.1	0.4	0.7	0.05	2	2.2	1	0.05
Cube (256 nodes)	0.1		0.014	1.8	0.0024	-	—	—	0.01
Mosaic (16K nodes)	0.0025	0.001	0.006	0.01	0.0036	—	—	—	0.001
Encore Multimax	138	15.5	67.3	609	12*	158	34.6	307	33.4
Warp	0.1			0.6	0.002		—	0.08	—
HBA									
16 nodes	3.2	0.1	0.17		0.55	–	–		
100+ nodes	0.6		0.37		0.91				

**24 bit fixed point

Tasks

- 1a-b Convolution, zero crossing
- 1c Output border list.
- 2 Connected Components.
- 3 Hough Transform.
- 4a Convex Hull.
- 4b Voronoi Diagram.
- 4c Minimal Spanning Tree
- 5 Visibility of Vertices
- 7 Minimum Cost Path

To Appear in: "The Characteristics of Parallel Algorithms"
L. Jamieson, D. Gannon, R Douglass Ed.
M.I.T. Press.

Measuring Communication Structures in Parallel Architectures and Algorithms

Steven P. Levitan

Department of Electrical and Computer Engineering
University of Massachusetts, Amherst, MA 01003

This work focuses on analyzing metrics for communication structures in parallel computer architectures. We review several measures or "metrics" of the interconnection networks of parallel architectures and we evaluate these metrics as predictors of machine/algorithm performance. We have done this by simulating six tasks on each of eight architectures and comparing the metric-predicted ranking of the machines/algorithms with the actual time complexity of the algorithms. We show that, in general, accepted metrics of machine architectures do not perform well as predictors of run-time performance. However, the performance of some parallel algorithms running on parallel machines correlates very well with the performance of other parallel algorithms on those same machines. We propose a set of tasks to be used as the beginning of a performance suite for evaluating the communication structures of parallel architectures.

1. Introduction

Parallel processing is not new. However, only recently have the interprocessor communication structures necessary to support parallel processing been recognized as a key issue in the design of parallel machines [50]. While there have been many attempts to classify and evaluate different communication structures [20, 36, 30], a real theory of parallel, or communication, complexity has not yet emerged [38].

Such a theory would allow computer architects to evaluate new ideas in parallel computer organization and perform comparative analysis between existing and proposed designs. It would give programmers a method for predicting the performance of parallel algorithms running on different parallel machines and a way to judge the performance of algorithms running on different architectures, in terms of optimality for each particular architecture. Just as the current theory of algorithmic complexity is a basis for algorithm design and analysis, an extension

of this theory to parallel algorithms and parallel machines will form a basis for parallel algorithm design and analysis techniques.

Many models for parallel computation have been proposed. Borodin and Hopcroft [9] give a hierarchy of models which could be extended to include The parallel comparison model of Valiant [54], Schwartz's Paracomputer [44] and Thompson's [51] and Vuillemin's [55] models of VLSI. Galil and Paul [21] also give a comparative analysis of models.

Models, however, are not real machines. Computer architects and programmers are concerned with how accurately models predict the abilities of real machines. Machines that can be built would approach the abilities of any model with a time penalty which is some function of the size of the system. How much of a time penalty is not always clear. For example, Schwartz [44] claims the Ultracomputer will be able to simulate the Paracomputer (and all simpler models) with only a factor of $\log(N)$ time penalty. He is using about $N * \log(N)^2$ hardware to perform routing between a set of N processors and N memory registers. In the notation we describe below, this is a $O_i(\log(N))$ instruction time penalty. This penalty is, itself, more than the running times of some of the algorithms we want to discuss. This makes it difficult to predict the running times of real algorithms on real machines.

The problem is that a model of a parallel computer for a theory of parallel complexity must account for the time penalties involved in communication. However, these time penalties are different for different parallel architectures. Since we clearly do not want a different theory of complexity for every parallel architecture, we must develop a theory which explicitly addresses the differences in communication structures of different architectures.

In an attempt to solve these and other problems, researchers have proposed several measures of communication structures. Some possible measures that have been proposed are: Diameter, Bandwidth and the Average Distance between every two processors [35], Message Capacity [26], Average Message Delay, Average Message Density, and Connections per Processor [58], the time to perform a Bitonic sort, the number of mapping functions performed by the network, the number of bijection functions performed by the network, and the time it takes for a network to simulate other networks [46,47].

Not all of these measures have been used in the context we are proposing, as a tool for complexity analysis, and not all of them would be appropriate for that purpose. More importantly, rather than some measure of the "raw power" of the structures, we are concerned with characterizations of the structures that will give us some insight into the way they will support the communication needs of real

algorithms. Therefore, the best measures are the ones that help us predict algorithm performance. Measures (or metrics) which predict algorithm performance could then be used in a general theory of parallel complexity.

There is no reason to believe that any of the proposed metrics are better than any others. Moreover, very little has been done to verify the quality of the metrics themselves. To address this issue, we have chosen a set of metrics to evaluate on the basis of how well they can be used to predict the abilities of different multiprocessors. The metrics we choose to evaluate attempt to capture those aspects of the multiprocessor communication structures which will directly affect algorithm runtime performance.

The research reported here consisted of several steps [33,34]. The organization of this paper follows our work. We first present the assumptions or "ground rules" for analyzing parallel machines and algorithms used throughout the work. We next select a set of multiprocessor architectures based on the communications structures of those architectures. We then consider a set of metrics for those structures which might be useful to capture their communications abilities. Next, we examine the runtime performance of several classic tasks on the different architectures. At the same time, we use the metrics to predict the performance of the machines on those different tasks. We then compare the runtime performance of the algorithms and the predictions of the metrics.

The rest of the paper evaluates how well the metrics can be used to capture the abilities of the architectures, and therefore predict their relative merits on different algorithms. The result is an evaluation of the metrics themselves, as tools to capture the nature of of machine/algorithm interactions. Besides simply evaluating the metrics as predictors, this work is also an examination of the way algorithms "fit" on different parallel architectures. We have tried to reveal the relationship of the communication needs of algorithms to the abilities of machine architectures. Finally, we conclude with a proposal for a set of benchmark routines or a performance suite of algorithms which we believe are more suitable for evaluating multiprocessor architectures.

Thus, the goal of the work is twofold: To characterize parallel algorithms in terms of their communication needs and to select and verify useful measures that can aid in the analysis of parallel algorithms. These characterizations and metrics are viewed as necessary precursors to a valid theory of parallel algorithm complexity.

2. The Model

We have chosen to use a version of the network model [41] for our research. We are trying to be "architecturally accurate." When we discuss machines, we would

like to eliminate as much as possible the hidden costs of converting these ideals to practical systems. We are doing this because we are seeking to evaluate metrics that will be used on real machines. As much as possible, we should consider architectures that can be realized directly. In particular, we are not considering any common-memory based architectures.

Our model assumes that the time for messages to travel along the arcs connecting processors is very small compared to the time to put the message on the arc, or to take it off at the other end. However, we are very restrictive about the abilities of the processors to handle messages. We charge one unit of time equal to the execution of an instruction for each "send" or "receive" operation.

More importantly, we do not assume that the processors can detect messages on their input arcs "for free." Each input must be tested which also takes a unit of time. In a hardware sense this means we are not allowing interrupts. Although hardware for arbitrating interrupts is well understood, the arbitration mechanism and the vectorization mechanism require hardware and gate delays, which grow as the system grows. We have decided to remove that complication from our analysis. As we show later, this means that our Fully Connected machine is weaker in our model than in other models. Routing is not a trivial problem since arriving messages are not immediately known at each processor.

2.1 Time Complexity of Communication and Computation

In addition to the instruction level delays for communication, we are also interested in considering the time for gate delays and wire propagation delays in a certain restricted sense. We are concerned with comparing operations that might take different amounts of time on different architectures. For instance we might need to contrast a single bit boolean OR operation on one architecture, with an arithmetic ADD on another. To allow for such "apples and oranges" comparisons, we need to have a consistent measure of how long operations take.

For serial machines the actual time to perform operations is not an issue and it is often consciously ignored by theorists. "An add is an add is an add," at least asymptotically. However, in large systems the time to move data around cannot be ignored. More importantly, when analyzing parallel algorithms we must contrast the time to move data between processors with the time to compute. Since we are talking about different degrees of sophistication of our processing elements, we must also separate out the difference in complexity between a "Gate Level" operation and a "Fetch Execute Instruction" level operation. As part of the model, there must be a way to compare different notions of computation with each other and with equally different implementations of communication.

There is no agreement about uniform time measures for communication costs

in parallel models of computation. Even the propagation delays in different types of wires have been dealt with differently in the past. For instance, fabricators of silicon circuits claim that the delay in silicon wires is intrinsically proportional to the length of the wire squared; wires to them are delay lines. Designers of integrated circuits however, use length as a rule of thumb. Using repeater circuits, Thompson [53] and Mead and Conway [38] claim a logarithmic delay. With other assumptions, researchers have different results. G. Bilardi, M. Pracchi, and F. P. Preparata [7] propose three different models for three special cases of signal propagation in VLSI silicon: $O(1)$, $O(L)$, and $O(L^2)$. They conclude that for most current technology we can use $O(1)$.

Of course, things are no better when we try to decide gate delay time. Is it a function of the number of inputs to a gate? Is it a function of the driven load? Is it a function of the technology? The answer to all these questions is "Yes." For operations, or instructions, we have no better solution. After all, machines are made out of wires and gates.

We propose the following resolution to the above problem. If we are considering CPU operations or instructions we use a subscript $O_i()$ (or none); if we are considering gate delays we use a subscript $O_g()$ and wire delays call for a subscript of $O_w()$. We use a similar convention for the lower bounds notation: $\Omega_i()$, $\Omega_g()$, $\Omega_w()$. We will use the three subscripts where appropriate in the analysis of our algorithms. This frees us to pursue our analysis without regard to technology-dependent conditions of the relationships between these domains. Also note, that we use $\log(x)$ to mean logarithm to the base 2 of x, $\ln(x)$ to mean the natural logarithm of x, and $x * y$ to mean $x \times y$.

3. Machines, Metrics, and Algorithms

For our work in testing metrics, we have chosen a representative sample of current or proposed parallel architectures. We consider one Single Instruction stream Multiple Data stream (SIMD) machine, the Content Addressable Parallel Processor. We consider five "graph-structured" machines of increasing complexity: the Star, Linear, Tree, Shuffle, and Full networks. We also include two unique designs of our own, the Broadcast Protocol Multiprocessor, and the Fully Interconnected with Content Addressable Parallel Processors machine. The machines have been selected on the basis of their broad range of communication characteristics.

We have chosen the metrics of Diameter and Bandwidth based on Lint [35], Path Count based on Horowitz and Zorat [26] Thickness based on Gannon [22], and our own measure, Narrowness [33]. These metrics span a range of complexity from simple measures to more sophisticated graph theoretic properties of the network.

The algorithms have been selected by three criteria. First, we have taken familiar, well studied, and well understood serial algorithms and considered them in the parallel environment. Second, we have limited the domain of our algorithms to combinatoric algorithms. These are integer arithmetic non-numerical algorithms generally concerned with decisions rather than numbers. Third, we have been concerned with sub-algorithms or "kernels." These are useful algorithms which occur often in larger programs.

3.1 Machines

The architectures we have chosen span the taxonomy presented by Anderson and Jensen [1]. We give the Anderson-Jensen classification for each machine along with a brief description.

3.1.1 The Content Addressable Parallel Processor (CAPP)

Single Instruction Multiple Data (SIMD) architectures [17] are based on the concept of a single central controller broadcasting instructions to all the processors in the machine. Each processor has its own data which it operates on in parallel with all the other processors of the machine. The Content Addressable Parallel Processor (CAPP) architecture is like a SIMD machine with simple processors [18]. The CAPP does not directly fit into the Anderson-Jensen classification (Figure 1).

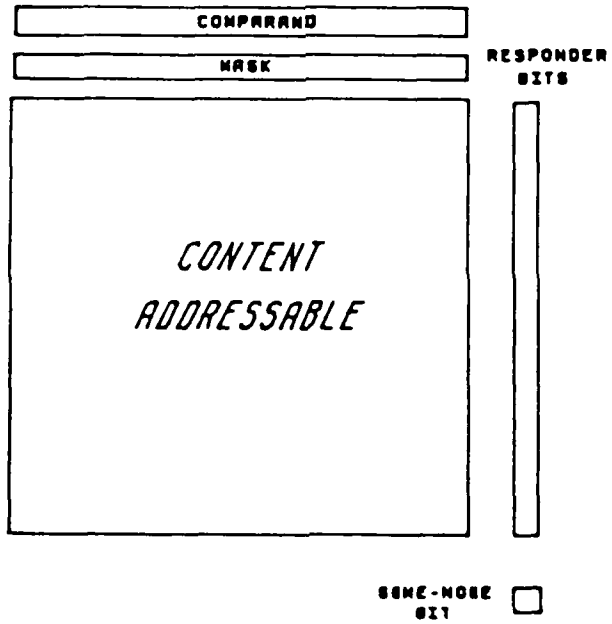


Figure 1. Content Addressable Parallel Processor (CAPP)

In a CAPP both instructions and data are broadcast. The processors, or processing elements, can operate on both their own internal data and data broadcast by the central controller. This allows for such operations as comparison of values, global updates of values, and arithmetic operations where one of the operands is a broadcast value. Each processor of the CAPP can save the results of these operations and conditionally execute future operations based on these results. The results are kept in boolean flag registers called "responder bits."

Another important feature of the CAPP is the ability of the processors to report their results back to the central controller. There are two types of summary reports. One is a simple existence result: Are there any processors with responder bits set? The second is a count of the number of processors with their responder bits set.

A design for a VLSI CAPP is proposed by Weems [56]. A survey of CAPPs and related architectures is provided by Foster [19] and Weems [57]. The CAPP is interesting primarily because of the broadcast and response circuitry. Also, it has reasonably little processing power at each "processor," but it is still a powerful parallel processor.

3.1.2 Star Network

The Star machine is made up of a group of processors (Spokes), each connected to a single central, or Hub, processor. The distance between any two processors is short (two) but the number of simultaneous messages that can be transferred is only one. This is an Indirect, Centralized Routing, Dedicated Path (ICDS) machine (Figure 2).

3.1.3 Broadcast Protocol Multiprocessor (BPM)

This machine is like a Star Network, except that the Hub is not a processor. Rather, the Hub is a single register that has the property that all processors can read its contents at once. All Spoke processors can attempt to write at once into the register, but only one (chosen at random) will succeed. This structure also models the communication abilities of networks like the Ethernet [39]. Details of this architecture are in [31, 32]. Depending on the central resource, this is a Direct, Shared Path Global Bus (DSB) or an Indirect Bus with a Central Switch (ICS) machine (Figure 3).

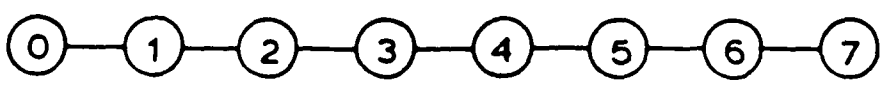


Figure 4. Linear Network

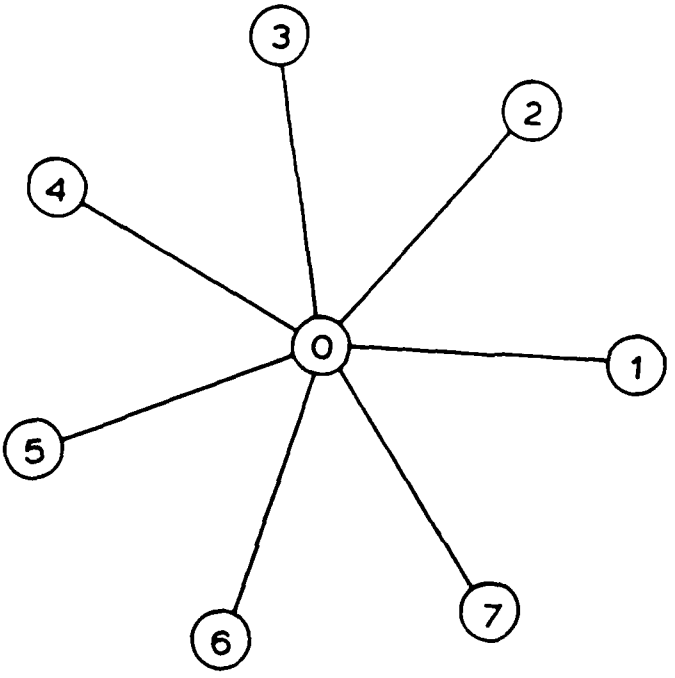


Figure 2. Star Network

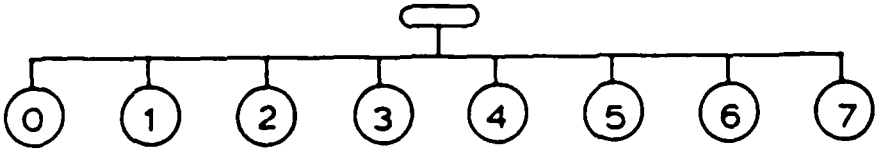


Figure 3. Broadcast Protocol Multiprocessor (BPM)

3.1.4 Linear Network

This machine is simply a group of processors interconnected in a linear fashion. There is a bidirectional link between a processor and its left and right neighbors. There is no "end around wrap." This architecture is interesting because it has a large distance between the two end processors, but the number of simultaneous messages in the network can also be large. This is a Direct, Dedicated Path Linear (DDL) machine (Figure 4).

3.1.5 Tree Network

This Machine is a group of processors connected in a binary tree. There is a distinguished Root and Leaves. This architecture has been proposed by several authors since the communication distance is shorter than it is for the Linear network and the number of simultaneous messages it can support is better than the Star network. This is an Indirect, Decentralized, Dedicated path, Irregular (IDDI) machine (Figure 5).

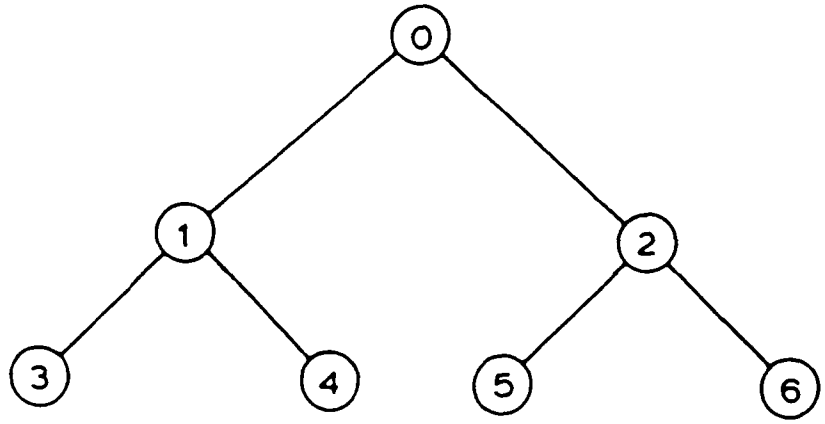


Figure 5. Tree Network

3.1.6 Shuffle Network

This interconnect pattern was originally proposed by Stone [49] as a method for interconnecting dynamic memories. In our machine, each processor $1 \ldots N-1$ is connected by four unidirectional links. Processor I can send to processor $(2 * I) \bmod N$, and processor $(2 * I + 1) \bmod N$. N must be a power of two. This can be thought of as a network where each processor is the root of a binary tree, connecting all processors. This is an Indirect, Decentralized, Dedicated path, Regular (IDDR) machine (see Figure 6 for numbering conventions).

3.1.7 Fully Interconnected Network (Full)

In this machine, every processor can communicate directly with every other processor. There are N^2 bidirectional links. However, each processor can send and receive only one message at a time. This is often considered the best possible interconnection scheme, even though it is expensive. This is a Direct, Dedicated path, Complete (DDC) machine (Figure 7).

3.1.8 Fully Interconnected with CAPP (Full with CAPP)

This machine has the feature that each processor has its own CAPP as part of

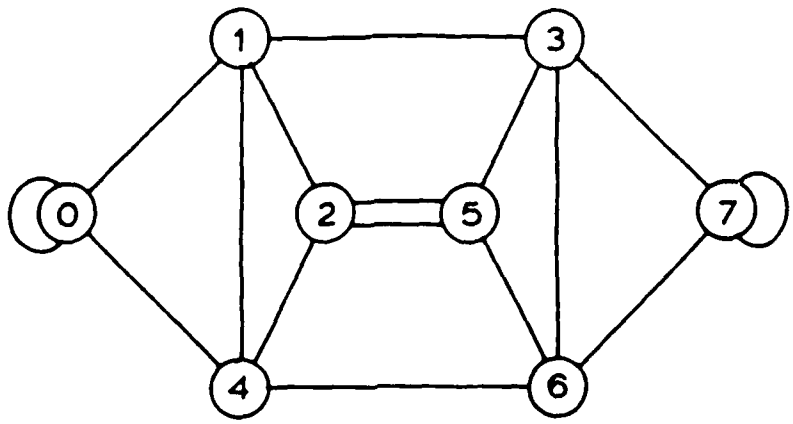


Figure 6. Shuffle Network

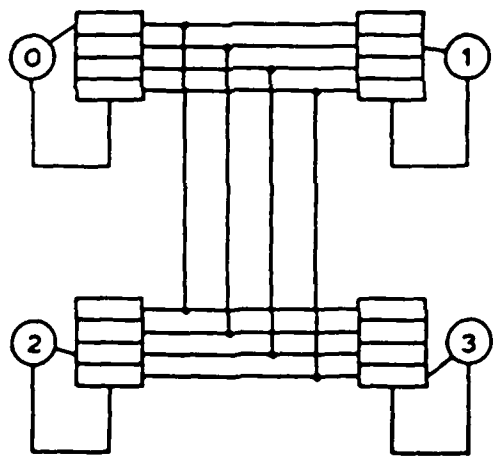


Figure 7. Full Network

its own memory. Furthermore, we let each processor's communication links end in a word of the CAPP of every other processor. This allows each processor to broadcast to every other processor in "unit" time (Figure 8).

The Full with CAPP machine is constructed by having each processor share a different word of CAPP with every other processor. This means that when one processor broadcasts to all words of its CAPP, the value appears in one word of

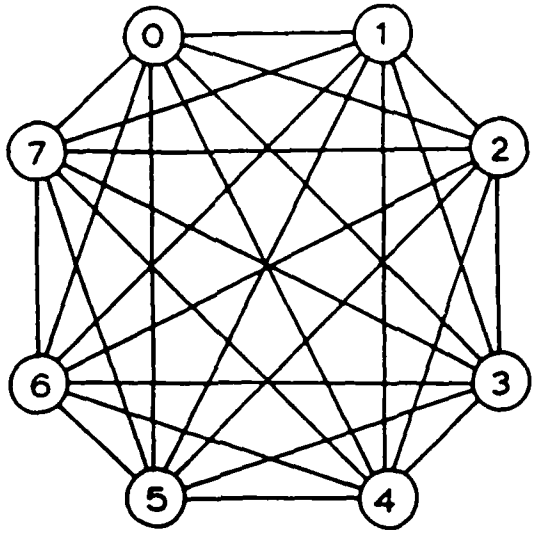


Figure 8. Full with CAPP Network

each of the other processor's CAPPs. Conversely, when one processor searches its CAPP it is really examining one word from each other processor's CAPP. Of the N^2 words $(W[i,j])$, each is written into by processor i and read and searched by processor j. This could be extended to allow both processors i and j to read and write the same word. However, to allow both processors to search the same word would imply N response bits for each word. Our algorithms do not need that capability.

The Full with CAPP architecture is useful in cases where a regular CAPP would need to serialize operations. CAPPs, being SIMD machines, need to serialize at every decision point in an algorithm. There is only one controller. If there is a branch point based on a data value in the words of memory, the controller needs to "turn off" some words, perform one case, then go back and deal with the other case. Our CAPP architecture has a stack of response store bits to facilitate that operation. This simplifies programming, but does not necessarily speed up the algorithms. For instance, in the sorting task a traditional CAPP needs to find the K-th largest value for $K = 1 \ldots N$. The operation is serialized since the CAPP cannot find both the 17th and 31st largest values at the same time. The Full with CAPP machine, having N controllers, can perform the ranking operation in parallel.

3.2 Metrics

Two relatively simple metrics that have been proposed for measuring the communications abilities of parallel architectures are Diameter and Bandwidth [35]. Diameter can be informally described as the worst case time for one message to go from one node of the network to some other node. Bandwidth is the maximum number of messages that can be sent and received in the network at once.

We want our definitions of Diameter and Bandwidth to capture our intuitive notions of "How fast can we get a message from one end of the network to the other?" and "How much communication can be going on at once?" We would also like the definitions to be both precise and general enough that we can apply them to different architectures than the ones we are considering in this work, without resorting to intuition for every special case. While these goals are easily realized for Diameter, we will see that it is more difficult to formulate a satisfactory definition of Bandwidth.

3.2.1 Diameter

We define Diameter empirically in terms of a set of experiments. These experiments proceed as follows: We have each processor in the network send a message to every other processor, one at a time. The worst case delay among the $N*(N-1)$ trials is the Diameter of the network. Delay is measured in units of receive-send (forwarding) time $O_i()$, gate delays $O_g()$, or wire length $O_w()$, as appropriate.

3.2.2 Bandwidth

There are two problems that arise when we attempt to formalize our intuitive notions of Bandwidth. The first problem is that there is a difference between the number of messages that can be active in a network and the number of messages that the processors can handle. The second problem is that we must decide whether to count broadcast messages with the same content as one message or several. Examining the behavior of infinite size networks with different size branching factors aids us in resolving both problems.

We first address the problem of the difference between the number of messages that could exist in the communication pathways of a network and the number that can be processed by the system. Consider the case of a message originating at the root of an infinite binary tree and passed down from parent to children such that it is broadcast to all nodes of the tree. The number of nodes, N, which have seen the message at any given step time, S, is the sum of the first S Fibonacci numbers, $F(S)$. The sum is itself a Fibonacci number: $F(S+2)-1$ [29]. The reason for this growth pattern is the forwarding rule of our model which implies that a processor can only forward one message at a time. In a binary tree the number of new nodes at each step is the number of nodes which are one step old, plus the number of nodes which are two steps old. Three-step old nodes have sent

messages to both their children and therefore do no further work. The growth is exponential. $F(S)$ is about equal to $\phi^S/\sqrt{5}$, where ϕ is the "golden ratio" defined to be $(1+\sqrt{5})/2$.

For a trinary tree the nodes stay active for three time steps, for a quaternary tree for four time steps, etc. For an infinitely branching infinite tree all nodes stay active. Therefore, each time step doubles the number of active nodes. For an infinitely branching tree, it is our forwarding rule which prevents us from concluding that an infinite number of messages could be sent throughout the network in the first few time steps. The number of active nodes $= 2^{S-1}$. Even in the fully connected architecture it takes eight time steps to broadcast a single number to 128 processors.

The preceding argument demonstrates why we believe that Bandwidth ought to be defined in terms of the number of messages that processors can handle rather than simply the number of messages that can be active in a network. In fact, it substantiates our intuitive notion that Bandwidth is not simply the maximum number of messages that could "live" in the network at any given time; but rather it must be some function of the number of messages that can be generated or received by the processors in the network at a given time. Under the definition of Bandwidth that we give below, the maximum possible value of the Bandwidth of an N processor machine is N.

We now turn to the second problem that arises when we attempt to formalize our definition of Bandwidth. Do we count copies of identical messages as one message or as individual messages? Put another way, do we count the number of messages sent, or the number received?

Again we turn to the extreme case of an infinite number of processors. If all processors are listening to an infinitely long broadcast bus, and one processor sends a message, then they all receive a copy of the message "immediately." We can say that there are an infinite number of messages received, or multiple copies of one message, or that the message moves down the wire with speed $O_w()$ and therefore takes infinite time to propagate. If we allow infinite buses to hold more than one message, we could claim that the messages need not even be multiple copies of the same original, and thus we could use the $O_w()$ propagation delay as a virtual buffer. We could even pipeline messages. The infinite broadcast bus does not resolve our dilemma.

If, however, we look at an infinite receive bus, a "wired OR" for example, only one message can be carried on the bus. It can be a message of consensus, a Some/None response, but it is only one datum of global information. The propagation of the message on this bus must be at least $\Omega_w()$, but it is only one

message. Therefore, for the purpose of symmetry (and simplification) we have decided to take the Bandwidth of both broadcasting and receiving buses as one message per unit time, $O_1()$.

Having considered the two problematic issues regarding Bandwidth, we are now able to state precise definitions for Diameter and Bandwidth. We define Diameter as the worst-case time to get a message from one processor of a network to another. We measure Diameter with the "time trials" experiment given above. We assume there is no other computation or communication in progress in the network at the time of the experiment.

We define Bandwidth as the total number of messages that can be sent or received by processors in the system in one unit of time ($O_1(1)$). We make the further restriction that a communication on a broadcast or receive bus counts as one broadcast, or receive, message respectively. Therefore, Bandwidth is the maximum number of unique messages that can be generated or consumed in unit time in a network.

The first two columns of table 1 gives the Diameter and Bandwidth of the architectures we are discussing. We can see that the machines are not completely differentiated based on these two metrics. The difference between the Star and the BPM is that the BPM uses broadcast messages, while the Star does not. The Tree and Shuffle networks differ in that the Shuffle has no root node with its associated bottle-neck. The difference between the Full and the Full with CAPP machines is that the Full with CAPP uses broadcast and report buses for its communications. In the next section we examine more sophisticated metrics which hope to capture these distinctions.

3.3 Message Capacity Metrics

Next we examine metrics for those structures that characterize how well the parallel machines support multiple message transfers between arbitrary processors in the network. We want to judge the ability of the network to handle the "typical" communication patterns that will be encountered during the execution of "typical" programs. Even though these metrics seem more general and less precise than Diameter and Bandwidth they do capture important properties of the networks and for some problems they can be used for deriving Lower Bounds. These "message capacity" metrics are based on an examination of message density along each of the interconnection paths between processors in the machine. They give a figure of merit for the message capacity of the network. There are several ways of calculating message capacity.

For comparison, we have chosen three different measures of message capacity. The measures are Path Count, Narrowness, and Thickness. Path Count can be

Table 1. Metrics of the Architectures

Machines	Metrics						
			. . . . . Paths				
	Diam.	Bndwdth.	Total	Avg.	W − C	Thick.	Narr.
Serial	1	1	1	1	1	1	1
Linear	N	N	N^3	N^2	N^2	1	N
Star	1	1	N^2	N	N^2	N	1
Tree	$\log(N)$	N	$N^2 \log(N)$	$N \log(N)$	N^2	1	N
Shuffle	$\log(N)$	N	$N^2 \log(N)$	$N \log(N)$	$N \log(N)$	N	1
BPM	1	1	N^2	N	N^2	1	N
CAPP	1	1	N^2	N	N^2	1	N
Full	1	N	N^2	N	N	N^2	$1/N$
Full/CAPP	1	N	N^2	N	N	N^2	$1/N$

thought of in terms of the number of paths through each node of the network, or equivalently the number of messages each node must handle in the task of each processor sending one message to every other processor. Narrowness measures the worst-case bottleneck which occurs when one partitions the network in all possible ways. Thickness is a refinement of Bandwidth; it is based on the task of each processor sending one message to one other processor in the "other half" of a partitioned network. The bisection width is minimized for all partitionings of the network into two parts, with half of the processors in each partition.

3.3.1 Path Counting Message Capacity

Our first measure is an extension of the ideas in Horowitz and Zorat [26]. They use the number of communication paths which each processor in a network is responsible for as a measure of the communication overhead in the system.

We define a Path in terms of the particular task of every processor sending one message to every other processor in the network. This generates $N * (N - 1)$ original messages. We make a distinction between original and total messages. For most machines, messages must be forwarded several times to get to their destinations. Each time a message is forwarded it is counted again.

We say there is a Path through a processor, X, if processor X is the source or destination of a message, or if processor X forwards a message from some processor, Y, to some processor Z. The number of paths through X is the number of messages (one to every other processor) that X creates, plus the number of messages (one from each other processor) that X terminates, plus the number of messages that X forwards for other processors, which are also sending one message to every other processor.

Using these rules we can calculate the total number of messages involved in the task of each processor sending to every other processor, and therefore calculate the number of paths that each processor must handle. We calculate the total number of messages, the average number of messages each processor must handle, and the number of messages the "worst-case" processor must handle.

3.3.2 Congestion based Message Capacity

Another measure, "Narrowness" [33] gives a more direct measure of the congestion to be expected in a network. The idea here is that for algorithms with lots of message traffic, the congestion dominates all other interactions in the system. Therefore, a measure which abstracts out that single property of the network is a good metric.

We calculate the Narrowness of the network as follows. We partition the network into two groups of processors A and B where the number of processors in each group is a and b respectively and assume $b \leq a$. Now we count the number of interconnections between A and B, call this i. We find the maximum value of (b/i) for all partitionings of the network. We call this measure the Narrowness of the network.

3.3.3 Worst-Case Bandwidth as Message Capacity

Gannon [22] uses a modified version of Bandwidth to give a figure of merit to networks. His work is an extension of the ideas used in VLSI complexity theory [52]. The basis of the measure is again that we partition the network, this time into two equal halves. Now we assume the underlying task is for each processor in one group to send a message to another processor in the other group. We compute the normalized time for this task. Gannon calls this the "Bisection Bandwidth." To avoid confusion we will call it the "Thickness" of the network.

Gannon shows how we can use this measure to give lower bounds for certain problems. Assume P is a problem for which parallel algorithm A provides a solution on a given architecture. If A needs to send k messages between two halves of the network during its execution and the Thickness of the network is T, then the communication part of the execution of A is bounded by k/T. If, in addition, the Numerical part of the execution of A has S steps and the network

has N processors, then the entire execution time is bounded by:

$$\text{Time} \geq S/N + k/T.$$

Gannon examines a class of algorithms based on "Divide and Conquer" with a Shuffle Exchange communication pattern. Of course, not all algorithms can be characterized by this exchange operation and determining k for other solution techniques is somewhat tricky. This does not negate the value of Thickness as a measure of machine communication capabilities. It is a valid measure of the different interconnection structures' abilities to support message passing.

We should note that for our machines, Thickness is proportional to $N/$ Narrowness. This is a consequence of the fact that the (b/i) value is maximized when the networks are partitioned in half. This is not generally true for all networks. Networks made up of tightly interconnected sub-networks, loosely interconnected, generally will not have their narrowest point at the middle of the network. Rather, it will be between some sub-network and the rest of the network.

3.4 Algorithms

The algorithms have been chosen to represent a broad range of complexity. They are also important "kernel" algorithms which occur as sub-problems in larger programming tasks. We explain our choice of algorithms and the algorithms themselves in the following sections. broadcasting, reporting, extrema, and packing are discussed first. Then we discuss sorting and the minimum spanning tree. We briefly give our motivation for selecting these algorithms and review some related work in the following sections.

We have deliberately chosen communication intensive problems for use in studying the metrics. This is despite the widely accepted belief that multiprocessors, in general, are more efficient when the ratio of communication to computation is low [30, 27, 24]. Jones, for example, calls the amount of computation done per communication the "grain size" of the tasks. She recommends that programmers try to keep the grain size of their parallel algorithms high. In other words, one would like to have the "semantic content" of the communications as high as possible. That way each message would be maximally useful to the completion of the total algorithm.

For our purposes in assessing the metrics, however, it is more informative to consider small grain size algorithms. There are two reasons for this. First, since our interest is on the effect of different network organizations on system performance, we want to "overload" the network as much as possible, i.e., stress the multiprocessor on the side of communication rather than computation. We have done this in order to clearly expose the impact of organization on performance.

Second, it is often possible to trade communication time for computation time, and in general that is the kind of tradeoff that will be done by creators of parallel algorithms. But in our analysis, we do not want the possible tradeoffs to complicate the issues of communication behavior. The simple algorithms were chosen expressly to keep such tradeoffs from obscuring the issues of communication behavior that are our primary concern.

The first four computational problems that we address in this section are relatively simple. Broadcasting is the problem of sending one value to all processors in the network. Reporting is the problem of determining if any processor has a particular value. Broadcasting and reporting are essentially pure communication algorithms; there is virtually no computation component to the algorithms. Maximum finding is finding the largest value from a set of items distributed one to a processor in the network. Packing is the problem of squeezing null values out of a sparse array of item values. The items are distributed throughout the network, and the algorithm must preserve the relative order of the non-null items. Maximum finding and packing involve slightly more computation than broadcasting and reporting, but they are still predominantly communications problems, with very minor computational components.

Sorting and computing the minimum spanning tree of a graph have been chosen as more complex tasks which also need a large amount of communication. For both these tasks our algorithms are based on initial conditions of each processor having one element of the input set. This might not always be the fastest way to compute the result, however we still want to use communication rather than computation whenever possible.

3.4.1 Broadcasting

Broadcasting is the task of sending a single message to all processors in the multiprocessor system. This is a basic operation for all multiprocessors. It is often provided by the hardware architecture itself, as in the case of the CAPP and the BPM machines. Even when it is not provided in hardware, it is a necessary function and must be somehow implemented to allow the processors to at least initiate and terminate tasks. Many parallel algorithms use broadcasting as a subalgorithm in their larger computation. For example, in several of the minimum spanning tree algorithms we need to tell every processor which node of the tree was chosen after each iteration of the algorithm.

For those machines that do not support broadcasting directly in hardware, the time that it takes to perform the operation is critical. As we see in later sections, it is often the most limiting aspect of the computation. Jordan [28] and Nassimi and Sahni [40] present algorithms for broadcasting in the Finite Element Machine and a SIMD square grid machine respectively.

3.4.2 Reporting

The purpose of reporting is the opposite of that for broadcasting. It is to gather information about the state of the network to a central location. In some cases the result is available to all processors as a side effect of the algorithm. The algorithm gathers one bit of information, reflecting existence of a particular value in the network. The demands the algorithm makes on the network are different than for broadcasting since there is some computation involved, logically OR'ing partial results before sending a new result on to its next destination. Even though this is a rather trivial amount of computation it imposes a synchronization constraint on the algorithm.

It is also worth noting that, aside from buses, Full interconnect structures are optimal for broadcasting problems. However, they are not optimal for fan-in tasks like reporting. For fan-in, binary (or trinary) tree structures are optimal [33]. Of course, the Full interconnect can just use the subset of its connections which make up a binary tree.

For broadcasting, the order that data flows around the network is irrelevant. Even redundant copies of the data value were not a problem. For reporting things are different. Each processor must wait for any partial values it is responsible for combining to arrive before it sends its results on. The delay due to synchronization is a coefficient on the time of the algorithm. Although it has no effect on the asymptotic running times of the algorithms, the difference is important.

3.4.3 Extrema (Maximum or Minimum) Finding

Extrema, or maximum, finding is the first "non-trivial" algorithm we explore. It is generally not available as a hardware operation in multiprocessors. It is an extension of reporting and can often be implemented as multiple calls to a report operation. It is also a useful sub-algorithm, particularly in algorithms where the multiprocessor is performing a distributed search operation. Each processor can speed up its local processing by having the best value found anywhere available for its local processing.

Maximum finding on multiprocessor architectures has been widely discussed. Hirschberg [25] and Chang and Roberts [12] discuss circular, or ring, architectures and give $O(N * \log(N))$ algorithms. Kung [30] describes algorithms on systolic arrays and trees. In trees, extremum can be found in $O(\log(N))$. Bokhhari [8] presents an $O(n^{2/3})$ algorithm for a square, n by n array augmented by a global bus. For a common memory model, Valiant [54] and Shiloach and Vishkin [45] present algorithms that have the surprisingly low running time of $O(\log(\log(N)))$. However, they do not discuss memory contention problems.

3.4.4 Packing

Packing is the task of moving data from higher numbered processors to lower numbered processors that have space in such a way that the relative order of the data among processors is unchanged. This is often used as a sub-algorithm to sorting and searching problems. As an example, parallel distribution sorts use this sub-algorithm. Packing is also a way to increase the locality of data in distributed processing algorithms. The multiprocessor can "shift down" related data so that the processors which have the data are physically near each other. Schwartz [44] presents packing on the Ultracomputer and on perfect shuffle machines.

Besides its applications, packing is interesting from a theoretical point of view. The average amount of data to be moved is less than that for sorting (or routing), and the average distance for data items to move is inversely proportional to the number of data items. Unlike routing, there is no way to know in advance where the data items are to end up since we do not know how many holes exist in the system. But, unlike Sorting, we do not have to do any comparisons since the data is already in order. The serial complexity of packing is $\Omega_s(N)$ since each value must be examined to see if, indeed, it is a hole.

There are several types of parallel algorithms for packing. The first are simply extensions of the serial algorithms. For both the Star Pack and the BPM Pack algorithms the Bandwidth of the network only allows for the disposition of one hole (or one non-hole) at a time. There are also "data migration" algorithms, where the active processors move data into non-active neighbors, or non-active processors ask for data from their active neighbors. Running times for these algorithms are dominated by the Diameter of the interconnect, as illustrated by the Linear Pack and the Tree Pack algorithms. Our Full Pack and Shuffle Pack algorithms are "swapping" based algorithms. Values and holes are exchanged in a pre-specified pattern based on the shuffle pattern.

3.4.5 Sorting

After examining extrema finding and packing, it is natural to examine sorting on a multiprocessor. Baudet and Stevenson [4] present sorting on linearly connected processors in $O(N)$ time. Kung, Valiant, and Shiloach all extend their discussions of Extrema Finding to include sorting. Trees and one dimensional systolic arrays can sort in $O(N)$. Shiloach sorts in $O(\log(N)^2)$ with common memory. Dewitt [15] and Thompson [53] survey results in parallel sorting. The most recent and complete bibliography on parallel sorting is in Richards [42].

3.4.6 Minimum Spanning Tree

The minimum spanning tree problem is interesting because it has several different solution techniques both for serial and parallel computers. Deo [14] gives several algorithms for a common memory architecture, the best of which has a time com-

plexity of $O(N*\log(N))$ for an N processor, N vertex system. Bentley [6] gives an $O(N*\log(N))$ algorithm for a tree machine of $N/(\log(N))$ processors. Savage [43] has a N^2 processor algorithm which runs in $O(\log(N)^2)$. Chang [10,11] and Yogen Dalal [13] give results for graph-structured parallel architectures, where the graph of the network is the graph for which the problem is to be solved.

There are several considerations which have a strong influence on how we can write parallel solutions for the MST problem. The first is a synchronization problem that occurs when we perform some of the operations in parallel. For instance, both vertices on an edge might have that edge as their minimum cost edge. Once one vertex claims that edge, the second vertex cannot use that edge too. Neither can it simply use its next best edge. The second best edge from that vertex might or might not be in the tree. Once two vertices share a common minimum edge, they must cooperate to find new edges. The general rule is that the edge that is picked must be the minimum of all edges of all vertices which have merged into a group, to a vertex which is not in that group.

A more subtle possible problem is that if we use a Sollin style "group merging" based algorithm, groups will merge in decreasing cost chains. Every vertex in all of the newly merged groups needs to know its "new" group identity so that it can evaluate its edges for the next merge. We call this propagation of group names across the network "contagion." The time to resolve the contagion is often the biggest time penalty for such parallel algorithms.

For our algorithms we also use the techniques of Prim, Dijkstra, Kruskal, and Sollin. The CAPP algorithm is the only one that uses the Kruskal style algorithm, and the Full with CAPP machine is the only one which uses the Sollin style algorithm. All the rest are variations of the Prim/Dijkstra algorithm. Our algorithms are for connected non-directed graphs. A summary of the running times of all the algorithms is presented in the next section.

4. Discussion

The details of the running of the algorithms, and their code is reported in [33]. The six problems: broadcast, report, max, pack, sort, and minimum spanning tree, on each of the machines: CAPP, Linear, Star, BPM, Tree, Shuffle, Full, and Full with CAPP, give a total of 48 parallel algorithms. These represent a fairly broad range of computational complexity which we have used for comparison purposes and as the basis of our analysis of the metrics.

The CAPP and Tree algorithms for broadcast, report, and max are based on standard algorithms for CAPPs and Tree Machines presented in Foster [19] and Bentley [5], respectively. The Linear Sort is based on Baudet and Stevenson [4]. The Shuffle Pack and Full Pack algorithms are based on Schwartz [44]. The

Shuffle Sort and Full Sort are based on Stone's mapping of the Bitonic Sort on the Shuffle-Exchange interconnect [48, 3]. The rest of the algorithms presented for these machines and all algorithms for the BPM, Star, and Full/CAPP machine are new.

We have also included a Serial machine and its times for reference. We define broadcasting and reporting for the serial machine as simply the act of performing a memory write or read respectively.

Some of the times in table 2 are given in terms of the value of the maximum value in the instance of the problem (Max). While this is accurate for the BPM, for the CAPP it is more correct to use the maximum possible value across all instances of the problem. Given that we are generally dealing with "almost N almost unique numbers," we will from now on take the value of N as the value of Max in our order relations. This follows the accepted practice of other researchers [53].

Table 2. Running Times of the Algorithms (order approx.)

Machines	Algorithms					
	Broadcast	Report	Extremum	Pack	Sort	MST
Serial	1	1	N	N	$N\log(N)$	$E\log(E), N^2$
Linear	N	N	N	N	N	N^2
Star	N	N	N	N	$N\log(N)$	N^2
Tree	$\log(N)$	$\log(N)$	$\log(N)$	$\log(N)$	N	$N\log(N)$
Shuffle	$\log(N)$	$\log(N)$	$\log(N)$	$\log(N)$	$\log(N)^2$	$N\log(N)$
BPM	1	1	log(Max)	N	N	$N\log(N), E$
CAPP	1	1	log(Max)	N	$N\log(\text{Max})$	E
Full	$\log(N)$	$\log(N)$	$\log(N)$	$\log(N)$	$\log(N)^2$	$N\log(N)$
Full/CAPP	1	1	1	1	$1^{\sim}$	$\log(N)^2$

4.1 Performance Summary of Diameter and Bandwidth

As we can see in table 2, some of the algorithm-machine performances are Bandwidth limited, some are Diameter limited, and some are simply not well

predicted by either Bandwidth or Diameter. We first discuss how Diameter and Bandwidth can be used to explain the behavior of the machines on the simpler algorithms for broadcast, report, max, and pack. Then we proceed to an evaluation of the other metrics and other algorithms.

The CAPP is Bandwidth limited for any operation which involves decision making. The central control must either "orchestrate" the decisions, as it does in maximum finding algorithms, or it must do the decisions serially, as it does for the packing algorithm. The Star machine, also, is severely Bandwidth limited on all operations. Unless we more fully use the processing power of each Spoke processor, it is essentially a Serial processor with N words of memory. Our algorithms were all picked to minimize the processing that each processor would do, so it is no surprise that we get this result. More complex algorithms do somewhat better on a Star.

The BPM has a low Bandwidth, but it has the advantage of global communications. We have defined Bandwidth as the number of unique messages in the system so it has a Bandwidth of 1. However, for broadcast and report the network as a whole is really only transmitting (or receiving) one message. So the BPM does well with its one global message capability. The Star also has a Bandwidth of 1 but it is not a "global one." For packing the BPM and the Star perform the same since these algorithms require $O_i(N)$ different messages.

There are no real surprises in the Linear network performance. It is as diameter-limited as the Star was bandwidth-limited. It also does better on more complex algorithms. The Tree is also Diameter limited. The interesting facts come from the coefficients of the order results on broadcasting and reporting. For broadcasting binary trees are not very good compared to N-ary trees; however, for reporting they are nearly optimal. For these first algorithms the Shuffle Network behaves much like the Tree because it really is designed to be a set of interlocking trees where each node is a root. We see this is a real advantage on algorithms with more data movement. The Full interconnect, as expensive in hardware costs as it is, does not do much better than the Tree and Shuffle networks. This is because we have defined send and receive operations to take one unit of time. As we argued before this is not an unreasonable assumption. The Full interconnect also does better on harder algorithms.

The Full with CAPP machine, of course, does quite well as a lot more hardware is dedicated to communication tasks than in any of the other machines. If we were to call the CAPP an N "processor" SIMD machine, then there are N^2 equivalent processing units in this machine. There are also N broadcast buses and N receive buses.

4.2 Diameter, Bandwidth and Time

We can see a relationship between Diameter and Bandwidth for several of our algorithms:

$$(\text{Diameter} / \text{Bandwidth}) * N = \text{time}$$

This is true for the Star, the Linear, the Tree, the Shuffle, and the Full with CAPP Machines for broadcast, report, max and pack.

This equation reflects the nature of the problems we have chosen to examine and the way we have defined Diameter and Bandwidth. For broadcasting there is no constraint on data movement or timing. For reporting and maximum finding there are slight synchronization restrictions. However, all of these functions have the properties of commutativity, associativity, and replication-insensitivity or "stability." By stability we mean that adding redundant copies of the input will not change the computed value.

Packing does not have as many of the nice properties as the other functions. Its running time on the Linear and Star machines is caused by their Diameter and Bandwidth restrictions respectively. The Tree algorithm is not really fair since we pack towards the root, which is not strictly a solution to the problem. Packing on the Shuffle network is done by taking advantage of its routing properties [48], which allows it to move data between processors that are far apart in $\log(N)$ steps without collisions or congestion. Packing on the Full with CAPP is so fast for two reasons. First, only relative positions are involved, and the tagged architecture removes the data dependency that comes from absolute addressing. Also, the ability of the Full with CAPP to perform unique operations on each data item is used.

The equation is not true for the CAPP, the BPM, and the Full Interconnect. The issue for the Full Interconnect is that each processor cannot examine all N inputs in unit time. For example, in a report algorithm where everyone sends to processor 0, even though the messages get to the processor in time $O_t(1)$, it cannot operate on its inputs fast enough. As we showed, the best we can do is "tree it." The Full with CAPP machine solves the problem by using N broadcast and N receive buses.

For an explanation of the times for the BPM and the CAPP we are brought up against our definition of Bandwidth. We have defined it to mean the total number of different messages in the system. Both of these machines use single global broadcast buses for all communication. The BPM has one for both broadcast and report, the CAPP has one for broadcast and another for report. This means that they will do better than the equation predicts for problems which involve only one global datum. For maximum finding they both use a radix based solution which

trades Bandwidth for Diameter. We use an SIMD algorithm which is feasible on networks with low Diameter. For packing both machines do no better than the Star, or Serial machines; this reinforces our intuition that their Bandwidth really is one.

This shows that a global bus is both more powerful and weaker than a full interconnect. For the case of non-unique messages that need to be sent, or formed by consensus, the bus is better. Some algorithms can take advantage of this multiple non-unique message capability. For cases where unique messages are needed, real point to point interconnect is clearly necessary.

For the BPM algorithms we can take advantage of the implicit synchronization imposed by the communication protocol [31, 32]. When there is an underlying ordering relation on the data, each processor can "make assumptions" on the values in other processors. These inferences can be based on what was *not* broadcast at a given time. We use this technique to sort with 100% efficiency; every value is mentioned exactly once in the process.

The BPM algorithms illustrate an important point about communication in multiprocessors. Communication and synchronization are really two aspects of the same coordination problem. In fact, synchronization is really just a special type of communication, where "control information" is sent rather than just "data."

Broadcast, report, max and pack are really not as communication intensive as other problems. These problems only require us to handle each data item once during the computation. Other problems, like sorting and the minimum spanning tree, often require that data items be moved and compared and moved again many times during the course of the algorithm.

4.3 Message Capacity Metrics

The rationale for our Message Capacity metrics is to capture the phenomena which lead to congestion and competition in network-based multiprocessors which cannot be described by Diameter or Bandwidth.

Diameter and Bandwidth are good metrics in the sense that they are easy to compute precisely and give hard bounds on performance. However, they do not always give tight lower bounds. Diameter gives bounds on the possible performance of algorithms which must move data from one "end" of the multiprocessor to the other. Bandwidth gives us a bound on how much data can be exchanged between processors during a computation. However, neither metric captures interference phenomena like the congestion near the root node in Tree networks. They do not predict well the running times for more sophisticated tasks like sorting.

This is not surprising. The information flow needed during sorting cannot be

simply characterized. Sorting is more than sending a message down the length of the network or collecting a datum from each processor. There must be patterns of exchanges: in some cases fixed, in some cases data dependent. The data movement itself must not be "serialized" by the constraints of the network. This is really the crux of the matter. The weakest link of the chain (or network) is the rate-limiting factor. By weakest we do not mean non-homogenous processors. It is their position in the network which makes different demands on their equivalent resources. As we see later, the Worst-Case Path metric is the one which best characterizes the requirements of sorting and the minimum spanning tree.

The Message Capacity metrics do capture the actual ability of networks to support high data traffic. However, they are not "fine grained" enough to really bound any particular algorithm. This is as much a result of our lack of understanding of the data movement pattern of algorithms as it is a shortcoming of the metrics.

5. Evaluating the Metrics

Evaluation of metrics is a subjective task. Different metrics are useful for abstracting different aspects of machine architecture, which in turn impose varying constraints for each algorithm. The values of the metrics were given in table 1.

The quality of the metrics as *predictors* of algorithm performance varies even more than their usefulness in establishing lower bounds. For instance, on the basis of Diameter alone we can set lower bounds for some algorithms on some machines; but Diameter tells very little about actual performance. Diameter is a very good predictor of algorithms on architectures which are "Diameter bound" like the Linear machine; however, it is a bad predictor of algorithms on other machines that have more severe constraints in their Bandwidth or Worst-Case Path Count.

We perform the evaluation of the metrics by defining a quantitative measure of how well the metrics can be used as predictors of algorithm performance. We rate the metrics by comparing their ability to rank our machines in the same order that the algorithms performed. We claim that this indicates how well they have captured the communication abilities of the machines to support real algorithms.

Our rating is done by a Rank Order technique [2]. First, we rank the machines by their performance on each algorithm. This is shown in table 3 where the machines are in "best-to-worst" order. We combine the broadcast and report rankings since they are the same. The horizontal lines indicate a change in running time, e.g., the Full-CAPP machine finds Extremum in $O(1)$ but the Full, Shuffle, Tree, CAPP, and BPM machines take $O(\log(N))$.

Next, we rank each machine by each metric as is shown in table 4. Again, horizontal lines indicate a change of value. We combine the Total and Average Path Counts and the Thickness and Narrowness rankings since these are the same. We use these two tables to generate our values of how well the metrics' ranking of the machines predicts the algorithm performance ranking of machines.

Table 3. Relative Ranking Machines by Algorithm Performance

Broadcast/Report	Extremum	Pack	Sort	MST
Full-CAPP	Full-CAPP	Full-CAPP	Full-CAPP	Full-CAPP
CAPP	Full	Full	Full	Full
BPM	Shuffle	Shuffle	Shuffle	Shuffle
Full	Tree	Tree	Tree	Tree
Shuffle	CAPP	CAPP	BPM	BPM
Tree	BPM	BPM	Linear	CAPP
Star	Star	Star	CAPP	Linear
Linear	Linear	Linear	Star	Star

Table 4. Relative Ranking of Machines by Metrics

Bandwidth	Diameter	Total/Average	Worst Case	Thick/Narrow
Full-CAPP	Full-CAPP	Full-CAPP	Full-CAPP	Full-CAPP
Full	Full	Full	Full	Full
Shuffle	Star	Star	Shuffle	Shuffle
Tree	CAPP	CAPP	Star	Star
Linear	BPM	BPM	CAPP	CAPP
Star	Shuffle	Shuffle	BPM	BPM
CAPP	Tree	Tree	Tree	Tree
BPM	Linear	Linear	Linear	Linear

We do this by giving points to each metric for correct predictions on the

ordering of each machine by each algorithm. We give each metric a point for each pair-wise relationship among machines that it predicts. In the rank order of the machines one is either "better than," "equal to," or "worse than" every other machine. This gives us $N * (N-1)/2$ possible relationships to get right; with eight machines a perfect score would be 28 for each algorithm.

As an example, the Thickness metric ranked the eight machines in the following order: Full-CAPP and Full, best at N^2; Shuffle and Star, N; then CAPP, BPM, Tree, and Linear, 1. To see how well Thickness predicts sorting we compare that ranking to the sorting algorithm rank of Full-CAPP, best at 1; Full and Shuffle, $\log(N)^2$; Tree, BPM, and Linear, N; CAPP and Star, $N * \log(N)$. We show this in table 5.

Table 5. Example Evaluation of a Metric

Thickness	Sorting	Points
Full-CAPP	Full-CAPP	6
Full	Full	5
Shuffle	Shuffle	4
Star	Tree	2
CAPP	BPM	1
BPM	Linear	0
Tree	CAPP	0
Linear	Star	0
	Total Points =	18

The points are computed as follows. Sorting ranked the Full-CAPP machine as better than all others; Thickness ranked it better than six machines but equal to the Full machine. It got six (out of seven) relations right, so we give Thickness six points on that line. For the next line, sorting ranked the Full machine equal to the Shuffle machine and better than the rest. Thickness ranked the Full machine as better than all six of the other machines. It got five (out of six) relations right, so we give it five points for that line.

On each line, we only look down the list so that we do not count relations twice. We accounted for the relation between the Full-CAPP machine and the Full machine on the first line so we need not consider it again.

Moving down the table we give four points for the fact that Thickness pre-

dicted the Shuffle machine would be better than the Tree, BPM, Linear and CAPP machines. It missed the relationship of the Star machine to the Shuffle machine. It gets two points for predicting that the Tree machine would do as well as the BPM and the Linear machines, and we give it one point for grouping the BPM machine with the Linear machine. It gets all the other relationships wrong.

The total score for the Thickness metric to predict the ability of our parallel architectures to sort is 18 out of 28. Carrying out the same calculation for each of the metrics and each of the algorithms gives us the results in table 6. For comparison, a Monte Carlo estimate of 10,000 trials between pairs of random rankings of eight items (each divided into three groups) gives an average value of 9.8.

Table 6. Comparative Evaluation of Metrics

Metrics	Algorithms						
	Broadcast/Report	Extrema	Pack	Sort	MST	*Average*	*Rank*
Diameter	16	12	10	7	10	11.00	4
Bandwidth	8	10	18	16	13	13.00	3
Total/Avg Path	14	10	8	7	8	9.40	5
W-C Path	10	14	20	20	16	16.00	1
Thick/Narrow	8	12	16	18	13	13.40	2

This rank-order based evaluation method has two nice properties. First, it only uses the relative values of both the metrics and the running times. We do not have to normalize any values. Second, it gives credit for "clustering" of the machines in the predictions. It does not penalize a metric too much for getting one relation totally wrong and thereby ruining the absolute order of the lists. For instance, if three machines were predicted to perform the same and they did, the metric gets credit for the "equal to" relations even if the rank of all three machines was wrong.

Table 6 illustrates several important points about the metrics. Diameter is a reasonable predictor of broadcast and report, but not very good at predicting the other algorithms. Bandwidth, on the other hand, is better at predicting the pack, sort, and MST algorithms. Both Total and Average Path Counts do poorly.

However, the Worst-Case Path Count is the best metric for predicting all the algorithms except broadcast and report. Thickness and Narrowness are tied for second place among the metrics.

The next to last column of table 6 shows the average prediction ability of each of the metrics. As above, a 28 would be a perfect score. The last column gives the relative rank of the metrics as predictors.

6. Conclusions

6.1 Algorithms as Metrics

Since broadcast and report are primitive to very many parallel processing tasks, we are lead to consider using the machine running times for these algorithms as metrics themselves. We have extended our comparison of metrics to include all our algorithms, as metrics for predicting machine performance (table 7).

We can see that, without exception, every algorithm is a better average predictor than any of the metrics in table 6. The average values in table 7 do not include the ability of an algorithm to predict itself. The minimum spanning tree algorithm is the best predictor. We believe this is because most of the programs written for the MST were made up of applications of the other algorithms as subtasks. In the next section we build on these ideas and propose a "performance suite" for parallel architectures.

6.2 A Performance Suite

In another age of Computer Architecture, architects designed "Business Machines" and "Scientific Data Processors" and pretended that those machines needed to be different. People soon discovered that the basic operations of machines were really the same no matter what "high level" operation they were doing. They Moved data, Operated on data, Branched, Saved and Restored state.

Even though the "operate on data" aspect of computer design can be very complex, it is conceptually a single function of the machine. People are rediscovering this phenomena with such architectures as the Reduced Instruction Set Computer (RISC) architecture [16].

This is not to say that machines should not support high level operations. Rather that even when they are supporting high level functions, they are doing it by the same basic operations listed above, no matter if it is done in macro-code, micro-code, or even "nano-code."

In this respect, multiprocessors are not so different from uniprocessors. Not only are each of the processors that make up a multiprocessor performing the basic operations, but the parallel processing machine as a whole is also performing those same operations. Here we are not talking about a random collection of computers

Table 7. Evaluation of Algorithms and Metrics as Predictors

Algorithms	Algorithms						
	Broadcast/Report	Extrema	Pack	Sort	MST	*Average*	*Rank*
MST	17	22	22	20	–	20.25	1
Pack	16	18	–	20	22	19.00	2
Extrema	20	–	18	15	22	18.75	3
Sort	12	15	20	–	20	16.75	4
Broadcast/Report	–	20	16	12	17	16.25	5
W-C Path	10	14	20	20	16	16.00	6
Thick/Narrow	8	12	16	18	13	13.40	7
Bandwidth	8	10	18	16	13	13.00	8
Diameter	16	12	10	7	10	11.00	9
Total/Avg Path	14	10	8	7	8	9.40	10

connected together, although we could even make the argument for that case, but a homogenous multiprocessor which is performing a single task.

Looked at in this context our work has simply expanded on the basic machine operation "Move Data" to reflect the possible complexity involved.

Following this thought, a set of "Universal" tasks should be appropriate to test the relative merits of multiprocessors. This set should contain a representative mix of communication patterns in much the same way that the Gibson [23] mix of operations is useful as a universal test for the "operate" group of instructions. This idea of a set of test cases is not new [37]; rather, it is time to extend these ideas to parallel architectures.

It is clear from our work that the fundamental operations of broadcasting and reporting should be included in this mix. Additionally, we have seen that sorting, as the general extension of routing, is also a fundamental operation. Our experience with packing and the minimum spanning tree has shown that not only is data routing important, but also data-dependent decisions need to be considered. This is reflected in the serializations caused by the dependencies in packing and the contagion in the minimum spanning tree. Although these two

tasks are not themselves universal, they do seem to encompass important aspects of interprocessor communication.

We propose the following tasks be used as a "Performance Suite" for evaluating the communication structures of parallel architectures:

- Broadcasting - - This is essential for any machine to support coordination of tasks. We need to broadcast simply to initiate most algorithms.
- Reporting - - Similarly, reporting is necessary for coordination and control.
- Selecting - - Maximum Finding or other "stable" functions which collect data from all nodes. Summing and other functions which are commutative and associative but not stable might be as good. The stable functions were easier to program, but their running times were not any faster than the corresponding non-stable functions would be.
- Sorting - - Routing and sorting reflect the machines abilities to support arbitrary communication patterns. This is the same as supporting all permutations.
- Propagating (Contagion) - - Packing or the Transitive Closure of a graph would all test the same abilities.
- Saturating (Many to Many) - - This is the task of each processor sending a message to every other processor. This measures bottlenecks or congestion in the machine.

There are many other possible tasks which could be included in such a mix besides these six. These are well documented in this paper and in the literature. We leave it to others to define the "ideal" set.

6.3 Future Work

As discussed in the Introduction, the ultimate goal of this research is to develop a theory of parallel complexity. We believe that a step towards that goal is a characterization of the communication structures of parallel architectures and the communication needs of parallel algorithms. Our goal has been to generate a set of metrics useful in this characterization. We have performed the exercise of solving a representative set of problems on a group of parallel machines and have discussed the techniques involved in writing those algorithms. We hope that this work will lay a foundation for describing the communication structures of parallel machines which is necessary for a usable theory of parallel complexity theory.

We propose that, until there is such a theory, an empirical approach is possible and desirable. This approach could be based on comparative studies of

machine performance on a suite of algorithms. This would allow us to continue our exploration of parallel architecture communication structures.

Acknowledgments

This work was supported, in part by AFOSR/DARPA grant F49620-86-C-0041. We would like to thank Richard Weiss, and Beverly Woolf for their help with drafts of the paper. We would also like to thank Caxton C. Foster for his ideas, support and guidance.

References

[1] Anderson, G. A., and Jensen, E. D., "Computer interconnection structures: taxonomy, characteristics, and examples," *Computing Surveys*, vol. 7, *4*, pp. 197-213, December, 1975.

[2] Andrews, T. G., *Methods of Psychology*. New York: John Wiley and Sons, 1978.

[3] Batcher, K. E., "Sorting networks and their applications," *AFIPS Spring Joint Computer Conference*, vol. 32, pp. 307-314, 1968.

[4] Baudet, G., and Stevenson, D., "Optimal sorting algorithms for parallel computers," *IEEE Trans. on Computers*, vol. C-27, *1*, January, 1978.

[5] Bentley, J. L., and Kung, H. T., "A tree machine for searching problems," CMU-CS-79-142, Dept. of Computer Science, Carnegie-Mellon University, Pittsburgh, Pennsylvania, August 30, 1979.

[6] Bentley, J. L., "A parallel algorithm for constructing minimum spanning trees," CMU-CS-79-142, Dept. of Computer Science, Carnegie-Mellon University, Pittsburgh, Pennsylvania, August 30, 1979.

[7] Bilardi, G., Pracchi, M., and Preparata, F. P., "A critique and an appraisal of VLSI models of computation," in H. T. Kung, B. Sproul, and G. Steele (Eds.), *VLSI Systems and Computations*. Rockville, Maryland: Computer Science Press, pp. 81-88, 1981.

[8] Bokhari, S. H., "MAX: An algorithm for finding maximum," *Proc. of the 1981 Int. Conf. on Parallel Processing*, Bellaire, Michigan, IEEE Computer Society, pp. 302-303, August 25-28, 1981.

[9] Borodin, A., and Hopcroft, J. E., "Routing, merging and sorting on parallel models of computation (extended abstract)," *Symposium on the Theory of Computing (STOC)*, pp. 338-344, May 1982.

[10] Chang, E. J.-H., "Decentralized algorithms in distributed systems," (Ph.D. Thesis), *Technical Report CSRG-103*, Computer Systems Research Group, University of Toronto, Toronto, Canada, October 1979.

[11] Chang, E. J.-H., "An introduction to echo algorithms," *Proc. First Int. Conf. on Distributed Computations*, Alabama, October 1-5, 1979.

[12] Chang, E., and Roberts, R., "An improved algorithm for decentralized extrema-finding in circular configurations of processors," *CACM, 22*, 5, pp. 281-283, May 1979.

[13] Dalal, Y. K., "Broadcast protocols in packet switched computer networks," *Technical Report No. 128*, Digital Systems Laboratory, Stanford Electronics Laboratory, Department of Electrical Engineering, Stanford University, Stanford, California, 94305, April 1977.

[14] Deo, N., and Yoo, Y. B., "Parallel algorithms for the minimum spanning tree problem," *Technical Report CS-81-072*, Computer Science Department, Washington State University, Pullman, WA, March 2, 1981.

[15] DeWitt, D. J., Friedland, D. B., Hsiao, D. K., and Menon, J., "A taxonomy of parallel sorting algorithms," *Computer Sciences Technical Report No. 482*, Computer Sciences Department, University of Wisconsin, Madison, Wisconsin, August 1982.

[16] Fitzpatrick, D. T., Foderaro, J. K., Katevenis, M. G. H., Landman, H. A., Patterson, D. A., Peek, J. B., Peshkess, Z., Sequin, C. H., Sherburne, R. W., and Van Dyke, K. S., "VLSI implementation of a reduced instruction set computer," in H. T. Kung, B. Sproul, and G. Steele (Eds.), *VLSI Systems and Computations*. Rockville, Maryland: Computer Science Press, pp. 327-337, 1981.

[17] Flynn, M. J., "Some computer organizations and their effectiveness," *IEEE Trans. on Computers*, vol. C-21, *9*, September 1972.

[18] Foster, C. C., *Computer Architecture*. (Second Edition), New York: Van Nostrand Reinhold, 1976.

[19] Foster, C. C., *Content Addressable Parallel Processors*. New York: Van Nostrand Reinhold, 1976.

[20] Freeman, H. A., and Thurber, K. J., "Updated bibliography on local computer networks," *Comp. Arch. News*, vol. 8, *2*, April 1980.

[21] Galil, Z., and Paul, W. J., "An efficient general purpose parallel computer,"

Symposium on the Theory of Computing (STOC), Milwaukee, Wisconsin., pp. 247-262, 1981.

[22] Gannon, D., "Notes on parallel algorithm taxonomy for numerical problems," Taxonomy of Parallel Algorithms Workshop, Los Alamos National Laboratory, Santa Fe, New Mexico, November 29 - December 2, 1983.

[23] Gibson, J. C., "The Gibson mix," TR00.2043, Systems Development Division, I.B.M. Corp., Poughkeepsie, NY, June 18, 1970.

[24] Gottlieb, A., and Kruskal, C., "Supersaturated ultracomputer algorithms," *Technical Report No. 024*, Department of Computer Science, Courant Institute of Mathematical Sciences, New York University, September 1980.

[25] Hirschberg, D. S., and Sinclair, J. B., "Decentralized extrema-finding in circular configurations of processors," *CACM*, vol. 23, *11*, pp. 627-628, November 1980.

[26] Horowitz, E., and Zorat, A., "The binary tree as an interconnection network: applications to multiprocessor systems and VLSI," *IEEE Trans. on Computers*, vol. C-30, *4*, April 1981.

[27] Jones, A. K., and Schwartz, P., "Experience using multiprocessor systems," *A.C.M. Computing Surveys*, vol.12, *2*, June 1980.

[28] Jordan, H. F., Scalabrin, M., and Calvert, W., "A comparison of three types of multiprocessor algorithms," *Proc. of the 1979 Int. Conf. on Parallel Processing*, Bellaire, Michigan, IEEE Computer Society, pp. 231-238, August 1979.

[29] Knuth, D. E., *The Art of Computer Programming Vol.3, Sorting and Searching*. Reading, Massachusetts: Addison - Wesley Publishing, 1973.

[30] Kung, H. T., "The structure of parallel algorithms," in M. C. Yovits (Ed.), *Advances in Computers*. Vol. 19, New York: Academic Press, pp. 65-112, 1980.

[31] Levitan, S. P., and Foster, C. C., "Finding an extremum in a network," *9th Annual Int. Symp. on Computer Architecture*, Austin, Texas, April 26-29, 1982.

[32] Levitan, S. P., "Algorithms for a broadcast protocol multiprocessor," *3rd Int. Conf. on Dis. Comp. Sys.*, Miami/Ft. Lauderdale, Florida, October 18-22, 1982.

[33] Levitan, S. P., *Parallel Algorithms and Architectures: A Programmer's Per-*

spective, Ph. D. Dissertation Computer and Information Sciences Department (COINS) Technical Report 84-11, University of Massachusetts at Amherst, May 1984.

[34] Levitan, S. P."Evaluation Criteria for Communication Structures in Parallel Architectures"; Steven P. Levitan; 1985 International Conference on Parallel Processing; St. Charles, Ill. August 20-23, 1985.

[35] Lint, B., "Communication issues in parallel algorithms and computers," Ph.D. Dissertation, University of Texas at Austin, TX. 1979.

[36] Lint, B., and Agerwala, T., "Communication issues in the design and analysis of parallel algorithms," *IEEE Trans. on Software Engineering*, SE-7, *2*, pp. 174-188, March 1981.

[37] Lucas, H. C. Jr., "Performance Evaluation and Monitoring", *Computing Surveys*, vol. 3, *3*, pp. 79-91, September 1971.

[38] Mead, C., and Conway, L., *An Introduction to VLSI Systems*. Reading, Massachusetts: Addison - Wesley Publishing, 1980.

[39] Metcalf, R. M., and Boggs, D. P., "Ethernet: distributed packet switching for local computer networks," *CACM*, vol. 19, *7*, p. 395-404, July 1976.

[40] Nassimi, D., and Sahni, S., "Data broadcasting in SIMD computers," *IEEE Trans. on Computers*, C-30, *2*, pp. 101-106, February, 1981.

[41] Preparata, F. P., "New parallel-sorting schemes," *IEEE Trans. on Computers*, vol. C-27, *7*, pp. 669-673, July 1978.

[42] Richards, Dana;"Parallel Sorting - a Bibliography"; SIGACT News, Vol. 18, No. 1; pp. 18-1.28, 18-1.48; Summer 1986.

[43] Savage, C., "Parallel algorithms for graph theoretic problems," Ph.D. Dissertation, Mathematics Dept., University of Illinois at Urbana-Champaign, Report ACT-4, Coordinated Science Lab., University of Illinois, August 1977.

[44] Schwartz, J. T., "Ultracomputers," *A.C.M. Trans. on Prog. Lang. and Sys.*, vol. 2, *4*, pp. 484-521, October 1980.

[45] Shiloach, Y., and Vishkin, U., "Finding the maximum, merging and sorting in a parallel computation model," *Technical Report 178*, Computer Science Department, Technion - Israel Institute of Technology, Haifa, Israel, March 1980.

[46] Siegel, H. J., "Analysis techniques for SIMD machine interconnection net-

works and the effects of processor address masks," *IEEE Trans. on Computers*, vol. C-26, *2*, pp. 153-161, February 1977.

[47] Siegel, H. J., "The universality of various types of SIMD machine interconnection networks," *Fourth Int. Symp. on Comp. Arch.*, March 1977.

[48] Stone, H. S., "Parallel processing with the perfect shuffle," *IEEE Trans. on Comp.*, vol. C-20, *2*, pp. 153-161, February 1971.

[49] Stone, H. S., "Dynamic memories with enhanced data access," *IEEE Trans. on Computers*, vol. C-21, *4*, pp. 359-366, April 1972.

[50] Sutherland, I. E., and Mead, C. A., "Microelectronics and computer science," *Scientific American*, vol. 237, *3*, pp. 210-229, September 1977.

[51] Thompson, C. D., "Area time complexity for VLSI," *Proc. 11th Annual ACM Symp. Theory Comput.*, pp. 81-88, April 1979.

[52] Thompson, C. D., "A complexity theory for VLSI," Ph.D. Dissertation, Computer Science Dept., Carnegie-Mellon University, Pittsburgh, Pennsylvania, August 1980.

[53] Thompson, C. D., "The VLSI complexity of sorting," *IEEE Trans. on Computers*, vol. C-32, *12*, pp. 1171-1184, December 1983.

[54] Valiant, L. G., "Parallism in comparison problems," *SIAM J. on Computing*, vol. 4, *3*, September 1975.

[55] Vuillemin, J., "A combinatorial limit to the computing power of VLSI circuits," *IEEE Trans. on Computers*, TC-32, *3*, pp. 294-300, March 1983.

[56] Weems, C. C., Levitan, S., and Foster, C., "Titanic: a VLSI based content addressable parallel array processor," *Int. Conf. on Computer Circuits*, New York, N.Y., September 29 - October 1, 1982.

[57] Weems, C. C., *Image processing with a content addressable array parallel processor*, Ph.D. Dissertation, Computer and Information Science Department, University of Massachusetts, Amherst, MA 01003, May 1984.

[58] Wittie, L. D., "Communication structures for large networks of microprocessors," *IEEE Trans. on Computers*, vol. C-30, *4*, pp. 264-273, April 1981.

END

5-87

DTIC